A Deployment Model for CORBA Components

Maria Claudia B. Barros, Edmundo R. M. Madeira, and Irineu Sotoma

Institute of Computing, UNICAMP – University of Campinas,
Av. Albert Sabin, 1251 – CEP 13083-970, Campinas, SP, Brazil
mcbb@terra.com.br {edmundo, isotoma}@ic.unicamp.br

1 Introduction

A software component has well defined interfaces and properties which facilitate the configuration and connection with other software components. By using components, the application developer only builds the business semantic and connects it to the existing components. The infrastructure to distributed communication, events, transactions, security, naming, and other services, are encapsulated in the components. By this way, it is easy to develop and maintain a business application.

The OMG (Object Management Group) has generated a component specification to CORBA (Common Object Broker Architecture) called CCM (CORBA Component Model) [2]. CCM provides support to definition, code generation, packaging, assembly and deployment of CORBA components to set up the entire server-side. The interesting point is that the CCM allows easy inter-working between CCM and EJB 1.1.

An important part of the use of components is the deployment [1]. The deployment of components consists in installation, instantiation and connection of components. However, the CCM specification does not detail the deployment implementation of CORBA components. So this paper proposes a model which extends CCM.

2 The Deployment Model

Our deployment model proposes a framework for component installation and activation. The main differences between our proposed model and the CCM deployment specification are: a class diagram which details the parties involved in the deployment process, a new repository called *Component Assembly Instance Repository* (CAI), a modified deployment process, and POA features for components. CAI stores the information of each component assembly instance deployed: component assembly identification, features of each component instance that are part of the assembly, the connections between components through events, and the connections between components through ports.

Fig. 1 shows the dependencies between the classes which implement the processes of the proposed model. The processes are: *Builder Tool*, *Populator of CAI*, *Deployment Manager*, *Installation Manager*, *Activation Manager*, *Activation Server*, *Event Server*, *Linking Manager*, and *Home Server*.

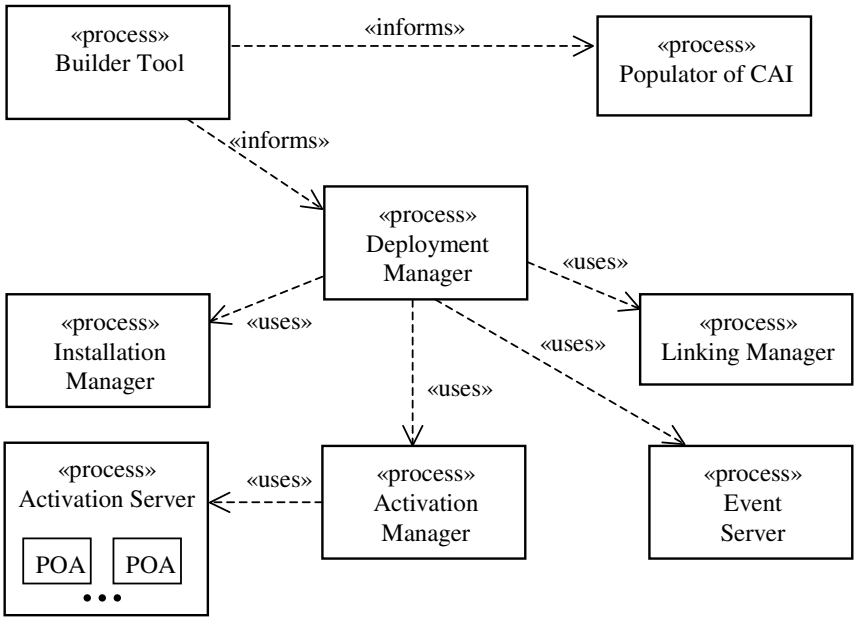

Fig. 1. Class diagram which shows the dependencies among the pieces of our *deployment architecture*

The *Builder Tool* shows a list of Components (based on Interface and Implementation Repositories) and gets the *user input* information. The *user input* information corresponds to: components which will form the user application, how these components will be connected and where they will be deployed. Afterwards, it generates an Assembly Archive (AAR). The *Builder Tool* also informs the *Populator of CAI* that the AAR file was generated. After the *Populator of CAI* has updated the CAI repository, the *Builder Tool* informs the *Deployment Manager* that it can start the deployment control. When the *Deployment Manager* completes the deployment, the *Builder Tool* returns to the user the `AssemblyIds` and `ComponentIds`.

The *Populator of CAI* reads the AAR file and creates an entry in the CAI that represents the assembly being deployed. The main responsibility of the populator is to generate: `AssemblyIds` (assembly unique identifier), `ComponentsIds` (component instance unique identifiers) and `ConnectionIds` (unique identifiers of the connections between components through events and ports). All the identifiers generated by the Populator are stored into the CAI repository.

The *Deployment Manager* coordinates the deployment process. It is activated by the *BuilderTool*, receives as input an `AssemblyId`, reads the necessary information from the CAI Repository, and interacts with the other managers and servers in order to create the deployment infrastructure in each machine. For each component instance it interacts with the *Activation Manager* to create the CORBA reference to the component. Then, it searches in the CAI entry for all fields that should contain the generated reference and updates them (in the *CompInstance* record, the field *Reference*, in

EventConnectionInstance records, the field *ConnectionSideARef* or *ConnectionSideZRef* (these records exist only if the component listens or generates events), in the *LinkConnectionInstance* records, the field *ConnectionSideARef* or *ConnectionSideZRef* (these records exist only if the component is linked to other component via a facet or a receptacle)). The *Deployment Manager* interacts with the *Event Server* in order to create the eventChannels for the component assembly and stores their references into the CAI Repository (in the *EventConnectionInstance* records the field *Reference*). The *Deployment Manager* interacts with the *Linking Manager* to "hook-up" components that are part of facets X receptacles connections.

The *Installation Manager* distributes the libraries that implement the components and their property files (if exist) to the machines where the components will be deployed.

The *Activation Manager* interacts with the *Activation Server* in each host requesting the creation of the references for the component instances being deployed in the host.

The *Activation Server* creates and maintains the POAs that generate the component instance references, and manages the lifetime of the servants that incarnate those component instances. For each component category and activation model a specific POA is created by the *Activation Server*. The activation model that better fits with components is On Demand: the POA creates the reference to the component but the servant will be created only at the first time this reference is used by a client. We use a Servant Manager to create the servants. This Servant Manager acts as a container, providing for the component servant access to the CORBA Services. The Servant Retention POA policy determines the Servant Manager behavior: Servant Location (`NON_RETAIN`) or Servant Activation (`RETAIN`). Another activation model useful is the Explicit, so at deployment time both the component reference and servant are created. This is an advantage for applications that get few components and/or need that the components are ready to be used immediately. The *Activation Server* is responsible for creating the POAs when it starts. At deployment time the user should choose an activation model (OnDemand-`RETAIN`, OnDemand-`NON_RETAIN` or Explicit) for each component instance. This information will be stored in the CAI Repository.

The *Event Server* is responsible for event channel creation and event producer and consumer "hook-up". The *Deployment Manager* searches in the CAI Repository entry for the pairs of components that communicate through events (consumers and producers) and decides the number and names of the event channels that will be created. The number varies according the algorithm being used, one possibility is to generate an event channel for each pair producer/consumer (simple approach), other possibility is to minimize the number of event channels being created reusing an event channel for the set of coincident producers and consumers. The name of the event channel is the concatenation of the *AssemblyId* and the *ConnectionId* to guarantee uniqueness. The *Deployment Manager* calls the *Event Server* (at a producer host) and asks for the *eventChannel* creation and the "hook-up" of producer(s) and consumer(s). The returned *eventChannel* reference is stored in the CAI (field Reference of *EventConnectionInstance* record(s)). Note that the fields *ConnectionSideARef* and *ConnectionSideZRef* of the *EventConnectionInstance* record(s) were previously filled in by the

DeploymentManager during the component instance reference generation, and are used by the *Event Server* to "hook-up" producers and consumers.

The *Linking Manager* links components through their facets and receptacles according to the information the *Deployment Manager* provides to it.

The user can consult the *Home Server* in order to obtain a reference to any component. The user calls the method `Home.resolve(assemblyId,componentId)`, which just gets previously created references from the CAI Repository that were created during the deployment process.

3 Conclusion

Because the deployment of CORBA components is an important issue in the use of components in applications, this paper proposes an extended deployment model which details the parties involved in the deployment process (repositories, processes, and user information). A proposed repository, the CAI, helps the deployment process by storing information about connections (through ports and events) between components.

In order to compare the different POAs for components, we have defined a simple component with one two-way operation. The component implementation was encapsulated in a DLL, and the POA request processing and the creation/binding/deletion of servants for the component were measured. To get the time elapsed between an operation invocation and its return, an ACE+TAO class was used.

According to performed experiments, this paper concludes the best performance of POA for components were obtained first by POAs with explicit activation model, which are 1.5 times faster in the first invocation and they are similar in the subsequent invocations, when compared to POAs with on demand activation model and `RETAIN` servant retention policy. The worst performance was obtained with POAs with on demand activation model and `NON_RETAIN` servant retention policy, which are 1.5 times slower in the first invocation and 5.65 times slower in the subsequent invocations, when compared to POAs with on demand activation model and `RETAIN` servant retention policy.

Acknowledgements. The authors thank FAPESP (Grant 00/05369-2), Pronex (SAE Project), and CNPq (Quaresma Project) by their financial support.

References

1. Kebbal, D., and Bernard, G. Component Search Service and Deployment of Distributed Applications. *Proc. 3rd International Symposium on Distributed Objects and Applications (DOA'01)*, Rome, Italy, September 17-20, 2001.
2. Object Management Group. CORBA Components Volume I. Document orbos/99-07-01. 1999.